

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開2001-142692

(P2001-142692A)

(43) 公開日 平成13年 5月25日 (2001.5.25)

(51) Int.Cl. ⁷	識別記号	F I	テームコード* (参考)
G 0 6 F 9/30	3 1 0	G 0 6 F 9/30	3 1 0 E
9/32	3 2 0	9/32	3 2 0 A

審査請求 未請求 請求項の数26 O L (全 22 頁)

(21) 出願番号 特願2000-278071(P2000-278071)

(22) 出願日 平成12年 9月13日 (2000.9.13)

(31) 優先権主張番号 0 9 / 4 1 1 1 4 0

(32) 優先日 平成11年10月 1日 (1999.10.1)

(33) 優先権主張国 米国 (U S)

(71) 出願人 000005108

株式会社日立製作所

東京都千代田区神田駿河台四丁目 6 番地

(72) 発明者 シバラム・クリシュナン

アメリカ合衆国、カリフォルニア州

94024、ロスアルトス、ウエストブルック・アベニュー 1723

(74) 代理人 100080001

弁理士 筒井 大和

最終頁に続く

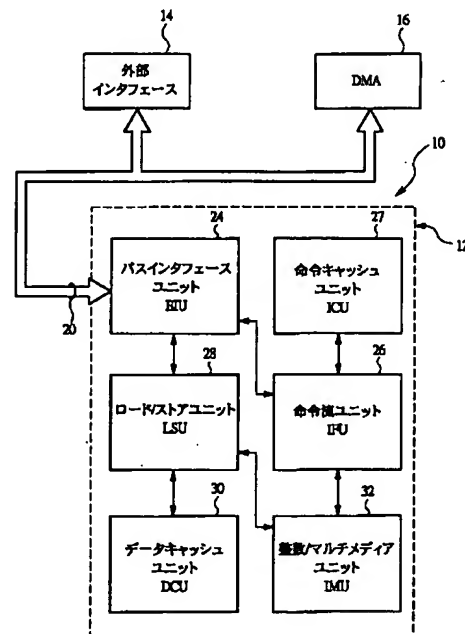
(54) 【発明の名称】 2つの異なる固定長命令セットを実行するマイクロプロセッサ、マイクロコンピュータおよび命令実行方法

(57) 【要約】

【課題】 32ビット固定長命令セットアーキテクチャを実行するように構成され、16ビット固定長命令セットアーキテクチャと後方互換性を持つプロセッサエレメントを提供すること。

【解決手段】 32ビット固定長命令セットアーキテクチャを実行するように構成されたプロセッサエレメント12は、各16ビット命令を1つ又は複数の32ビット命令のシーケンスに変換することによって16ビット固定長命令セットアーキテクチャと後方互換性がある。16ビット命令実行と32ビット命令実行との間のスイッチングは目標命令が16ビット命令または32ビット命令のどちらであるかを識別するために分岐の目標アドレスの最下位ビット位置を用いる分岐命令によって達成される。

図 1



【特許請求の範囲】

【請求項1】 MおよびNが整数であってMがNより小さい場合に、Mビット命令セット又はNビット命令セットから操作可能なプロセッサであって、それぞれNビット命令およびMビット命令を有する少なくとも第1および第2命令ストリームを記憶するメモリユニットと、

Nビット命令を実行するために実行信号を受け取るように操作可能な実行ユニットと、

実行信号を生成するために前記メモリユニットから前記第1および前記第2命令ストリームを受け取って復号するように前記メモリユニットおよび前記実行ユニットに結合された復号ユニットであって、前記復号ユニットが、前記復号ユニットによる復号を実施するために、各々の前記Mビット命令を受け取って前記Mビット命令の第1グループの各々を前記Nビット命令の対応する1つに変換し、前記Mビット命令の第2グループの各々を2つ以上のNビット命令へ変換する変換ユニットを含む復号ユニットと、

からなるプロセッサ。

【請求項2】 前記Mビット及びNビット命令の各々が、Mビット命令のメモリアドレスを識別するための第1状態およびNビット命令のメモリアドレスを識別するための第2状態に設定された少なくとも1つのビット位置を有するメモリアドレスによって識別される前記メモリユニット内の場所に記憶される請求項1に記載のプロセッサ。

【請求項3】 前記Nビット命令ストリームが少なくとも1つのNビット分岐命令を含み、実行中のNビット命令から実行中のMビット命令へスイッチするために前記Nビット分岐命令を実行するように前記デコードユニットが作動する請求項2に記載のプロセッサ。

【請求項4】 Nが2Mに等しい請求項1に記載のプロセッサ。

【請求項5】 Mが16であり、Nが32である請求項1に記載のプロセッサ。

【請求項6】 1つのビット位置が前記メモリアドレスの最下位ビットである請求項2に記載のプロセッサ。

【請求項7】 プロセッサユニットであって、

複数の命令を記憶するメモリであって、MとNが整数でMがNより小さいMビット命令およびNビット命令を含み、各命令がメモリアドレスによって識別された記憶場所に記憶され、各メモリアドレスが前記Mビット命令に関する第1状態および前記Nビット命令に関する第2状態へ設定された1つのビット位置を備えてなるメモリと、

検索された命令の実行を制御するためのメモリから前記命令を検索するための命令流制御ユニットであって、前記命令流制御ユニットが、2つ以上の前記Nビット命令のシーケンスに変換するために、前記Mビット命令の幾

つかを受け取るように操作可能な変換ユニットを含む命令流制御ユニットと、

からなるプロセッサユニット。

【請求項8】 マイクロプロセッサであって、

MとNが整数であってMがNより小さい場合に複数のMビット命令およびNビット命令を含むメモリエレメントと、

前記Mビット命令またはNビット命令の選定された幾つかを検索するために前記メモリエレメントに結合された命令フェッチユニットとを有し、前記命令フェッチユニットは、

前記メモリエレメントからフェッチされた前記Mビット命令の各々を1つ又は複数のNビット命令のシーケンスに変換するトランスレータユニットと、

前記メモリエレメントからフェッチされたNビット命令の各々及び前記トランスレータユニットからNビット命令の各々を受け取って、この種のNビット命令を復号するために前記メモリエレメント及び前記トランスレータユニットへ結合される復号ユニットとを含む、

マイクロプロセッサ。

【請求項9】 目標アドレスを保持するための少なくとも1つの目標レジスタを含み、複数の前記Nビット命令が、前記マイクロプロセッサによって実行される時に目標レジスタに目標アドレスをロードする準備目標命令を含む請求項8に記載のマイクロプロセッサ。

【請求項10】 前記複数のNビット命令が、それぞれ変換または復号するために1つのMビット目標命令または1つのNビット目標命令を前記メモリエレメントからフェッチさせるように分岐するために前記目標レジスタ内の前記目標アドレスを使用するように作動するBLINK分岐命令を含む請求項9に記載のマイクロプロセッサ。

【請求項11】 前記BLINK分岐命令が無条件分岐命令である請求項10に記載のマイクロプロセッサ。

【請求項12】 前記目標アドレスがMビット目標命令を識別するために第1状態に設定された1つのビット位置を含む請求項9に記載のマイクロプロセッサ。

【請求項13】 前記ビット位置がNビット目標命令を識別するために第2状態に設定される請求項12に記載のマイクロプロセッサ。

【請求項14】 MとNが整数であってNがMより大きいMビット命令およびNビット命令を記憶するメモリと、

受け取った各Mビット命令を1つ又は複数のNビット命令のシーケンスに変換するためのMビット命令を受け取るために前記メモリに結合されたトランスレータと、前記Nビット命令を受け取って復号するために前記メモリおよび前記デコードに結合されたデコードとを含む、単一チップ上に形成されたマイクロコンピュータ。

【請求項15】 Mが16であり、Nが32である請求

項14に記載のマイクロコンピュータ。

【請求項16】 前記Nビット命令が目標命令の分岐アドレスを表示するデータを含むNビット分岐命令を含み、前記目標命令がMビット命令である時に第1状態に設定された1つのビット位置を前記分岐アドレスが有する請求項14に記載のマイクロコンピュータ。

【請求項17】 MとNが整数であってMがNより小さい場合に単一チップ上に形成されたマイクロコンピュータによってMビット命令およびNビット命令を実行する方法であって、

前記Mビット命令および前記Nビット命令をメモリに記憶するステップと、

前記Nビット命令の幾つかを順次復号するように第1モードにおいて操作するステップと、

前記Mビット命令の幾つかを1つ又は複数のNビット命令のシーケンスに順次変換するように第2モードにおいて操作し、次に、前記Nビット命令を復号するステップと、

を含む命令実行方法。

【請求項18】 前記Nビット命令が1つのNビット分岐命令を含み、前記第1モードにおいて操作するステップが、操作の前記第1状態から第2状態へスイッチするために第1状態に設定された最下位ビットを含むメモリアドレスを有する1つのMビット命令へ分岐するように前記Nビット分岐命令を復号するステップを含む請求項17に記載の方法。

【請求項19】 Nビット分岐命令を含むNビット命令、及び、Mビット分岐命令を含むMビット命令を実行するように構成されたマイクロコンピュータにおいて、MとNが整数であってNがMより大きい場合に、Mビット命令を実行する方法であって、

1つ又は複数のNビット命令のシーケンスでMビット分岐命令の各々をエミュレートするステップと、分岐アドレスを提供する準備分岐命令で、その後で、前記分岐アドレスを使用するNビット分岐命令でMビット分岐命令をエミュレートするステップと、

を含む命令実行方法。

【請求項20】 少なくとも1つの目標アドレスレジスタを提供するステップを含み、前記Mビット分岐命令をエミュレートするステップは、前記目標アドレスレジスタに目標アドレスをロードする準備分岐命令を含む請求項19に記載の方法。

【請求項21】 前記Mビット分岐命令をエミュレートするステップが、前記Nビット分岐命令が前記目標アドレスを読み取るステップを含む請求項20に記載の方法。

【請求項22】 各々が1つ又は複数の前記Nビット命令にตอบสนองしてデータを記憶するための複数の汎用レジスタを含む請求項14に記載のマイクロコンピュータ。

【請求項23】 複数の前記汎用レジスタの各々が2N

ビット位置を含む請求項22に記載のマイクロコンピュータ。

【請求項24】 1つ又は複数の前記Nビット命令の中の所定個数のNビット命令が、複数の前記汎用レジスタの中の選定されたレジスタの低位ビット位置にデータをロードする請求項22に記載のマイクロコンピュータ。

【請求項25】 1つ又は複数の前記Nビット命令の中の選定された第1のNビット命令が、複数の前記汎用レジスタの中の選定された1つの低位ビット位置にデータをロードし、1つ又は複数の前記Nビット命令の中の選定された第2のNビット命令が、複数の前記汎用レジスタの高位ビット位置にデータをロードする請求項22に記載のマイクロコンピュータ。

【請求項26】 1つ又は複数のNビット命令の中の所定個数のNビット命令が、複数の前記汎用レジスタの中の選定されたレジスタの高位ビット位置において複製されたデータの最上位ビットのエキステンションと共に複数の前記汎用レジスタの中の選定されたレジスタの低位ビット位置にデータをロードする請求項22に記載のマイクロコンピュータ。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は一般にマイクロプロセッサ/マイクロコントローラアーキテクチャに関し、詳細には、第2の更に小さい固定した命令への後方互換性を備えた第1の固定長命令セットを実行するように構成されたアーキテクチャに関する。

【0002】

【従来の技術】電子産業におけるミニアチュア化およびパッケージングの分野における最近の進歩は様々な「埋め込まれた」製品の設計に関して機会を提供している。埋め込まれた製品は一般に小さくかつ手持ち式であり、制御機能用のマイクロコントローラ又はマイクロプロセッサを含むように構成される。埋め込まれた製品の例には、例えばセル電話、ポケットベル（登録商標）、及び、個人用デジタル補助装置（PDA）のような手持ちビジネス用、消費者用、および、工業用デバイスが含まれる。

【0003】埋込み設計すなわちアーキテクチャを成功させるには、例えば、埋め込まれる部品のサイズ及び電力消費のような或る種の必要条件を考慮しなければならない。この理由に因り、埋め込まれた製品用の或る種のマイクロコントローラ及びマイクロプロセッサは、命令の比較的小さいセットの迅速かつ効率的な処理に焦点を絞った縮小命令セット計算（RISC）アーキテクチャを組み入れるように設計されている。ただし、初期のRISC設計は32ビットの固定長命令セットを使用した。処理エレメントを更に最小限化するために、例えば16ビットのような小さな固定サイズを使用する設計が開発され、コンパクトなコードの使用を可能にし、命令

メモリのサイズを縮小した。小さくコンパクトなコードと結合したRISCアーキテクチャは埋め込まれた製品の設計が更に簡潔、小型、電力重視的であることを可能にする。この種の16ビットアーキテクチャの一例は米国特許第5,682,545号に開示されている。

【0004】

【発明が解決しようとする課題】しかし、16ビット命令セットによって提供可能であるよりも更に大きな計算能力および融通性の必要性が存在し、特にグラフィックスに対する能力が要求される時にそのような必要性が大きくなる。この必要性を満たすために、32ビット命令セットアーキテクチャが利用可能にされつつある。ただし、この種の32ビット命令セットアーキテクチャを用いると、一層大きい32ビット命令を記憶するための一層大きいメモリサイズが必要とされる。更に大きいメモリサイズは、埋め込まれた製品の設計の成功に逆行する傾向のある更に高い電力消費およびより多くの空間に関する必要条件の必要性を伴う。

【0005】更に、現在の32ビット命令セットアーキテクチャは、該当する場合には、初期に開発された16ビットコードに対する後方互換性を殆ど提供しない。その結果として、実質的なソフトウェア投資は失われる。従って、従来の小さい方のコードを使用するアプリケーションは廃棄するか、又は、32ビット命令に対してコンパイルし直さなければならない。

【0006】従って、サイズ及び電力消費拘束条件に殆ど影響を及ぼすことがないと同時に、初期の命令セットアーキテクチャへの後方互換性を提供する32ビット命令アーキテクチャを提供する必要性のあることが分かるはずである。

【0007】

【課題を解決するための手段】本発明は、概して、大きい方の固定長命令セットアーキテクチャ又は初期に設計された小さい方の固定長命令セットアーキテクチャのどちらでも実行するように構成され、これによって小さい方の命令セットへの後方互換性を提供する例えばマイクロプロセッサ又はマイクロコントローラのようなプロセッサエレメントに向けられる。小さい方の命令セットの実行は、主として、大きい方の命令の1つ又は複数のシーケンスを用いて小さい方の各命令をエミュレートすることによって達成される。更に、小さい方の命令セットアーキテクチャの資源（例えば、レジスタ、状態ビット、および他の状態）は大きい方の命令セット環境の資源にマップされる。

【0008】本発明の一実施の形態において、大きい方の命令セットアーキテクチャは32ビット固定長命令を使用し、小さい方の命令セットは16ビット固定長命令を使用する。ただし、当技術分野における当業者は分かるはずであるように、2つの異なる命令セットは任意の長さであっても差し支えない。16ビット命令の第1グ

ループは、それぞれ、1つの単一32ビット命令シーケンスによってエミュレートされる。16ビット命令の第2グループは、それぞれ、2つ以上の32ビット命令のシーケンスによってエミュレートされる。当該分岐の目標が一方の命令セット（例えば16ビット）のメンバーであることを識別するために所定の状態、又は、当該目標がもう一方の命令セット（32ビット）のメンバーであることを識別するために反対の状態にセットされた1つのビット位置（好ましい実施形態における最下位ビット（LSB））を持つ目標アドレスを使用する分岐（分岐）命令により、実行のモード間のスイッチングが達成される。

【0009】特定の16ビット命令セットアーキテクチャは分岐命令に関するいわゆる「遅延スロット」を含む。遅延スロットは分岐命令に密接して後続する命令であり、分岐命令の或る態様の準備期間中であって、当該分岐（分岐）が実施される以前に実行される（分岐命令がそのように指示する場合）。この様式において、分岐に関するペナルティが軽減される。遅延スロット命令を伴う16ビット分岐命令をエミュレートすることは、目標レジスタをロードする分岐命令に先行する準備分岐（PT）命令を用いて達成される。次に、分岐命令は、分岐用目標レジスタの内容を使用する。ただし、遅延スロット必要条件付き16ビット命令をエミュレートする時は、分岐は実行されるが、16ビット遅延スロット命令のエミュレーション及び実行が完了するまで、目標命令（分岐が実施される場合）は一時停止状態に保持される。

【0010】32ビットPT命令は、分岐目標プロセッサエレメントの通知を分岐命令から分離することによって、32ビット命令セット環境において低ペナルティ分岐を提供するように作動する制御流れ機構の一部を形成する。これは、プロセッサハードウェアが先行する多くのサイクルに互る分岐に気づかされることを可能にし、現行命令シーケンスから目標シーケンスへの円滑な遷移を可能にする。更に、それは、分岐ペナルティを最小限化するために16ビット命令セットアーキテクチャにおける遅延スロット技法使用の必要性をなくする。

【0011】本発明の特徴は、16ビット命令または32ビット命令のどちらでも使用することにより、各々の長さが64ビットの幾つかの汎用ないし多目的レジスタを提供する。ただし、汎用レジスタは、低位の32ビットが使用される場合に限り、16ビット命令によって書込みまたはロードされる。更に、符号ビットの自動エクステンションは、大抵の16ビット命令が汎用レジスタをロードする時に実施される。即ち、64ビット汎用レジスタの低位ビット位置に置かれた32ビット量の最上位ビットが、当該レジスタの32の高位ビット全てにコピーされる。32ビット命令セットアーキテクチャは、16ビットと32ビット環境の間に互換性を提供するた

めにこのプロトコルを使用するように構成された命令を含む。

【0012】同様に、64ビット状態レジスタは16ビット命令セット及び32ビット命令セットの両方に装備されている。状態レジスタの所定ビット位置は、16ビット命令セットからマップされる状態用に予約される。他の16ビット状態は汎用レジスタの所定ビット位置にマップされる。16ビット命令セット状態のこのマッピングは、タスク切り替えに必要な全ての文脈をセーブするために、個別環境(16ビット、32ビット)を可能にし、32ビット命令を用いた16ビット命令のエミュレーションを容易にする。

【0013】多数の利点が本発明によって達成される。16ビットコード及び32ビットコードの両方を実行する能力は、プロセッサが普通のタスクのためにコンパクトな16ビットコードを使用することを可能にする。次に、これは、両メモリスペースの節減および当該節減を伴った他の利点(例えば、更に小さいメモリ、節減された電力消費等)を可能にする。更に複雑なタスクが必要とされるときは、32ビットコードが使用できる。

【0014】更に、初期に設計された16ビット命令セットアーキテクチャを実行する能力は、当該初期設計においてなされた投資の保持を可能にする互換性を提供する。

【0015】PT命令は、分岐に関する事前通知を提供することにより、分岐命令の遂行におけるより多くの融通性を可能にする。

【0016】本発明のこれらの及び他の利点および特徴は、添付図面と共になされる以下の詳細な記述を読むことにより、当該技術分野における当業者にとって明白になるはずである。

【0017】

【発明の実施の形態】本発明は好ましくは既に開発済みの16ビット固定長命令セットアーキテクチャに後方互換性を提供する。この種アーキテクチャの更に完全な記述は「SH7750 Programming Manual」(プログラミングマニュアル)(Rev. 2.0、版権1999年3月4日)(Hitachi Semiconductor (America) Inc., 179 East Tasman Drive, San Jose, CA 95134から入手可能)に記載されている。

【0018】さて、図面、特に図1を参照することとし、ここに本発明の教示に従って作成されたプロセッサエレメント(例えばマイクロコンピュータ)の概略ブロック図を示す。図1に示すように、一般に参照番号10によって識別されるプロセッサシステムは、プロセッサエレメント12、外部インタフェース14、及び、システムバス20によって相互接続された直接メモリアkses (DMA) ユニット14を含む。外部インタフェース

14は、外部メモリに接続するように構成されることが好ましく、他の処理エレメント(例えば周辺デバイス、通信ポート、等)への通信的アクセスをプロセッサエレメント12へ提供しても差し支えない。

【0019】また、図1は、プロセッサユニット12を外部インタフェース14及びDMA16とインターフェイスするバスインタフェースユニット(BIU)24をプロセッサエレメント12が含む状態を示すことによって、プロセッサエレメント12の論理的配分も図示する。外部インタフェース14を介してシステムバス20及び外部メモリ(図示せず)に対するおよびこれらからの全てのリクエストを処理するBIU24は命令流れユニット(IFU)26へ通信的に接続する。IFU26は、命令キャッシュユニット(ICU)27からフェッチする命令を復号し、命令復号および実行パイプラインのフロントエンドとして貢献するように作動する。図に示すように、IFU26は、本発明に従って16ビット命令セットに32ビット命令セットのシーケンスをエミュレートさせる翻訳(ないし変換)論理を含む(以後、16ビット命令セットアーキテクチャを「モードB」と称し、32ビット命令セットアーキテクチャを「モードA」と称することとする)。

【0020】同様に、BIU24は全てのメモリ命令を処理し、データキャッシュユニット(DCU)30の操作を制御するプロセッサエレメント12のロード/ストアユニット(LSU)28へ接続する。整数/マルチメディアユニット(IMU)32は、全ての整数およびマルチメディア命令を処理するためにプロセッサエレメント12に含まれ、プロセッサエレメント12用の主データバスを形成する。

【0021】IFU26は、主要部分において、プロセッサエレメント12のシーケンサとして機能する。その主要機能は、ICU27から命令をフェッチし、それらを復号し、レジスタファイル50(図2)からオペランドを読み取り、復号済み命令およびオペランドを実行ユニット(IMU32及びLSU28)へ送り、実行ユニットから結果を収集し、それらをレジスタファイルへ書き戻す。その上、外部メモリ(図示せず)からの紛失命令で命令キャッシュを満たすことに命令キャッシュが失敗すると、IFU26はメモリリクエストをBIU24に発出する。

【0022】IFUの別の主要タスクはモードB命令のエミュレーションを実施することである。詳細には、特定のモードB命令がモードA命令の1つ又は一連のモードA命令のどちらかによってエミュレートされるように全てのモードB命令が翻訳される。次に、モードA命令は、元のモードA命令意味論へ、極めて小さい変更を伴って実行される。この方法は、モードB命令を実施するために必要な回路および論理が少数の機能論理ブロック内において分離されることを可能にする。これは、将

来、モードB命令セットの変化を可能にするか、又は、おそらく更に重要には、モードB命令セットを完全に除去することを可能にする利点を持つ。

【0023】図2はIFU26の構成を一層詳細に示すブロック図である。プロセッサエレメント12内における順序付けはIFU26によって行われるので、IFUは、プロセッサエレメント12の他の殆ど全てのユニットとインターフェイスされている。IFU26とBIU24及びICU27両ユニットとの間のインタフェースは、ICU27への命令のローディングおよび実行に関するICU27からの命令の流れを処理するICACHE (命令キャッシュ) コントロール (ICC) 40によって確立される。ICU27とLSU28及びIMU32の間のインタフェースは、命令、オペランド、結果、ならびに、命令の実行を可能化する全ての制御信号の送信/受信経路を供給する。これらのインタフェースに加えて、同様にIFU26は、外部割込みをサンプリング及び調停する外部割込みコントローラ41から外部割込み信号も受け取る。次に、IFU26は外部割込みと内部例外を調停し、非同期イベントを取り扱う適当なハンドラを作動化する。

【0024】図2に示すように、ICC40は、アクセスを準備するために、内部的にはフェッチユニット (FE) 42と、外部的にはICU27と交信する。通常、FE42は1つの命令フェッチアドレス、及び、ICC40に「フェッチデマンド」を表示する1組の制御信号を供給する。復帰に際して、ICC40は、最大2つまでの語整列命令ワードをFE42に送り返す。ICU27がミスする時には、ミスしているキャッシュラインに外部メモリ (図示せず) からロードするために、ICC40はBIU24への補充サイクルを開始する。補充は、FE42が元のフェッチアドレスを保持している期間中に行われる。その代りに、FE42は、命令の返却を要求しない「プリフェッチリクエスト」、あるいは、キャッシュミス (欠落) が起きた時に一切の補充活動が必要としない「フェッチリクエスト」を供給しても差し支えない。

【0025】FE42によってICU27からフェッチされた命令は、命令の命令セットアーキテクチャモード (即ち、モードB又はモードAのどちらか) に従って先ずバッファエリア42aに貯蔵される。ただし、最終的に、命令は、復号 (DEC) ユニット44への適応のために、2つの命令バッファの1つに移送されるはずである。

【0026】プロセッサエレメント12がモードA命令を実行しているとき、DEC44は命令を復号し、復号済み命令情報をFE42、分岐ユニット (BR) 46、及び、パイプラインコントロール (PPC) 48へ、また、外部的にはIMU32及びLSU28へ送るはずである。情報は、当該命令をそれ以上復号することなく、

LSU28及びIMU32がデータオペレーションを開始することも可能にする。分岐命令に関しては、部分的に復号された分岐情報は、出来るだけ早期に分岐方向を静的に予測するために、BR46を作動可能化する。

【0027】モードB命令が実行中であるときには、全ての命令は追加パイプラインステージ、即ちDEC44のモードBトランスレータ44aを通過するはずである。モードBトランスレータ44aは、各モードB命令を、1つまたは複数のモードAエミュレート命令に変換するはずである。次に、モードAエミュレート命令はDEC44のバッファまで移動させられ、ここで、正常モードA命令復号および実行が再開する。一例として、ここに記載される付録Aは、モードB移動および算術命令の各々に関して、モードB命令をエミュレートするために用いられるモードA命令シーケンスを示す。(既に確認済みのSH7750プログラミングマニュアルにおいて見られるように、モードB命令セットには、浮動小数点命令を含む更に多くの命令が含まれる。付録Aは、エミュレーションを示すためにのみ用いられる。) 当技術分野における当業者は、エミュレーションシーケンスが特定の命令セットアーキテクチャに依存することを認識するはずである。

【0028】更に、32ビットのデータの処理に際して互換性を保証するために、追加モードA命令は、モードB (32ビットデータ) 命令をエミュレートするためのモードA命令セットに含まれる。付録Bに示すこれらの追加命令は、当該命令において識別されたソースレジスタの下から32ビットのみを検索することによって32ビットデータを処理するように操作する。この操作のあらゆる結果は、当該命令において識別済みの宛先レジスタの下から32ビットに書き込まれ、書き込まれた量の符号ビット (即ち、最上位ビット) は宛先レジスタの上から32ビットに拡張されるはずである。

【0029】モードA命令によるモードB命令のエミュレーションの一例は、付録Aに記載されたモードB加算 (ADD) 命令によって示される。これは、1つの単一モードA命令である1つの加算長 (add. 1) 命令によってエミュレートされるモードB命令の1つである。モードB命令セットアーキテクチャにおいて、ADD命令は、2つの16汎用レジスタRm、Rnの内容を相互に加算し、汎用レジスタRn内の結果を記憶する。(理解されるであろうように、16個の汎用レジスタ (R0-R15) は、64ビット汎用レジスタ (R0-R15) 50の低位の32ビットにマップされる。) このモードBのADD命令のエミュレーションは、汎用レジスタの下から32ビットのみを用いるモードA加算長 (add. 1) 命令を使用する。Add. 1は、汎用レジスタRmの内容を汎用レジスタRnの内容に加えるように操作し、汎用レジスタRnの下から32ビットにおける結果を符号ビットの自動エクステンションと共に当該レジ

タの上から32ビットに記憶する。それによって、同一タスクを実施し、同一32ビット結果を得るために、モードBのADD命令はモードAのadd. 1命令によってエミュレートされる。(モードA命令は汎用レジスタの64ビット全体を使用する。レジスタに記入されるべき値が64ビット全体より少ないならば、モードB命令またはモードA命令のどちらかによって記入される場合であって、大部分が符号なし演算の場合であってもその値の符号は当該レジスタの高い方のビット位置に拡張される。これは、モードBまたはモードA演算の結果が64ビットの結果を生成するとみなされることを可能にする)。モードB命令セットに関して、上述のSH7750 Programming Manual (プログラミングマニュアル)に記載されているように、加算済みモードA命令は、ここに示す付録Bには記載されていない。

【0030】一連の2つ以上のモードA命令によるモードB命令のエミュレーションの一例はモードB桁上げあり加算(ADDC)命令により付録Aに示される。レジスタR_m、R_nの内容が符号なし数として扱われること、及び、モードB命令セットアーキテクチャの1ビットTレジスタに記憶されている以前の加算によって生成された桁上げが和に含まれることを除けば、ADDC命令はADD命令と同じである。ADDCが桁上げを生成する場合には、ならば、その桁上げは、後続するADD命令による使用のため又は他の演算のためにモードB環境において1ビットTレジスタに記憶される。これは、一連のモードA命令によるエミュレーションを必要とする。(レジスタに関しては図2のレジスタファイル50に含まれる64ビット汎用レジスタを参照されたい。)理解可能であるように、ADDC命令は一連のモードA命令6個によってエミュレートされる。

【0031】1. モードA加算符号なし長(addz. 1)命令は汎用レジスタR₀の下から32ビットを汎用レジスタR₆₃(定数「0」)に加え、その結果を、当該結果の上から32ビットに拡張されたゼロと共に、スクラッチパッドレジスタとして用いられる汎用レジスタR₃₂へ戻す。

【0032】2. 次に、addz. 1は、汎用レジスタR_nの下から32ビットを汎用レジスタR₆₃に加え、その結果を、当該結果の上から32ビットに記入されたゼロと共に汎用レジスタR_nへ戻す。

【0033】3. 次に、モードA加算命令はR_nとR₃₂との内容を相互に加算し(両者共、下から32ビット位置に32ビット量および上から32ビット位置にゼロを有する)、その結果をレジスタR_nに記憶する。

【0034】4. モードA加算命令は、現在レジスタR_n内に在る結果を、より早く生成されて、汎用レジスタR₂₅のLSBに置かれている桁上げに加算し、その結果をレジスタR_nへ戻す。

【0035】ステップ4の結果は1ビットTレジスタに設定されている桁上げを生成することがあり得るので、モードB環境において、Tレジスタがマップされているレジスタ(汎用レジスタR₂₅のLSB)は、エミュレーションの残りのステップに際して、桁上げと共にロードされる。

【0036】5. R_nに保持された値は、加算によって生じたあらゆる桁上げを当該値のLSBへ移動させるために、右方へ32ビット位置シフトされ、その結果はR₂₅に書き込まれる。

【0037】6. 最後に、レジスタR_nの内容は符号拡張された値ではないので、モードA即時加算命令は内容をゼロに加え、符号拡張された結果をR_nに戻す。

【0038】同様に、当該命令によって用いられるレジスタに記憶済みの値に応じて単一モードA命令または一連のモードA命令2個にエミュレートされたモードB命令がある。この二重特性エミュレーションの例は3移動(Move)データ命令であり、この場合のソースオペランドはメモリ(ソースが@R₀である場合にはMOV. B、MOV. W、MOV. L)である。モードB環境において、これらの命令は、汎用レジスタR_mの内容によって指定された記憶場所においてメモリ内データを検索し、それをレジスタR_nの内容に加え、その結果をレジスタR_nに戻す。次に、mがnに等しくないならば(即ち、データは、当該メモリアドレスを保持するレジスタ以外の全ての他のレジスタに移動中である場合)、レジスタR₀の内容はインクリメントされる。付録Aに見られるように、データが、当該データのメモリアドレスを保持する汎用レジスタへメモリから移される場合には、ただ1つの命令が用いられる。そうではなくて、データが他の場所に移動中であれば、当該メモリアドレスは第2の命令によってインクリメントされる。

【0039】図2へ戻って、BR46は全ての分岐に関する命令を扱う。BR46は復号済み分岐命令をDEC44から受け取り、分岐条件および目標アドレスが既知であるかどうかを決定し、分岐を解決/予測するために進行する。分岐条件が未知であれば、BR46は分岐条件を静的に予測する。次に、予測された命令はフェッチされて、復号される。場合によっては、予測済み命令がフェッチされ、分岐条件が解決される前に復号されることもあり得る。これが起きると、予測が正しいことをBR46が確認するまで、予測済み命令は復号ステージに保持される。

【0040】BR46は、8個の目標アドレスレジスタ46a、ならびに、状態レジスタ(SR)46bを含む幾つかの制御レジスタを有する(図3)。分岐は、部分的に、目標アドレスレジスタ46aの1つ又はもう一方の内容に基づいて処理される。特定の目標アドレスレジスタには、当該分岐の準備中に到来する分岐命令に先つ任意の時点において、分岐準備(PT)命令の使用によ

り、目標アドレスの書き込みが可能である。更に十分に検討されるように、分岐の事前準備に目標アドレスレジスタ46aを使用すれば、当該分岐のペナルティが軽減される。

【0041】(SR)46bは、実行の現行スレッドによって実行される命令の動作を制御するためのフィールドを含む制御レジスタである。SR46bのレイアウトを図3に示す。「r」フィールド(ビット位置0、2-3、10-11、24-25、29、32-63)は予約済みビットを示す。簡潔に述べれば、本発明に係るSR46bのフィールドは次のように作用する。

【0042】1ビットフィールドS、Q、M(各々のビット位置は1、8、9)は本発明の理解に関連しない特定の算術演算に際してモードA命令によるモードB命令のエミュレーションに用いられる。これらのビット位置は、モードA命令セットアーキテクチャによりモードB命令をエミュレートする際に用いられる、モードB命令セットアーキテクチャ環境からマップされた状態である。

【0043】1ビットフィールドFR、SZ、PR(各々ビット位置は14、13、12)はモードB浮動小数点命令の追加演算コード認定を供給するために用いられる。

【0044】モードB命令セットアーキテクチャは、なにかんずく、符号なし加算演算から得られる桁上げビットを維持するためにも1ビットTレジスタを使用する。モードBのTレジスタは、上述したように、汎用レジスタR25のLSBにマップされる。他のマッピングを以下に説明する。ただし、特定のマッピングが、エミュレートされる特定の命令セットアーキテクチャおよび当該エミュレーションを実施する命令セットアーキテクチャに依存することは当該技術分野の当業者によって理解されるはずである。

【0045】一旦、DEC44によって命令が復号されると、PPC48は、例えばLSU28、及び/又は、IMU32のような残りのパイプステージを介してそれらの実行を監視する。PPC48の主要機能は、命令が正しくかつ円滑に実行されること、及び、(1)全てのソースオペランドが、必要に応じて(IMU32乗算累積内部伝送のために)、準備が整っているか、又は、準備を整えることが可能になるまで、命令がデコード復号ステージ段階に保持されること、(2)命令ならびに全ての内部/外部事象によって課せられる全ての同期化および直列化必要条件が維持されていること、(3)全てのデータオペランド/一時的結果が正しく伝送されていること、を保証することである。

【0046】PPC48の制御論理を簡素化するために、モードA命令セット実行に関する幾つかの観察が仮定が行われる。これらの仮定の1つは、どのIMU命令も、決定論的に、例外を起こすこと、及び、全ての流れ

がバイパスステージを通過するようにすることは不可能であるということである。この仮定は、PPC48が、IMU32を、入力オペランドの源および出力結果の行き先を知る必要のない複雑なデータ演算エンジンとして見ることを可能にする。

【0047】PPC48の他の主要機能は、例えば命令例外、外部割込み、リセット等のような非順次的事象を処理することである。正常な実行条件の下において、PPC48のこの部分は常にアイドル状態にあり、事象が発生するとき、目をさます。PPC48は、外部の割込みコントローラ(図示せず)から外部割り込み/リセット信号、及び、プロセッサエレメント12の多くの部品から内部例外を受け取る。どちらの場合にも、PPC48はパイプラインを清掃し、コア状態および分岐をセーブするためにBR46に、また、分岐を適切なハンドラに通知する。複数の例外および割込みが同時に発生する時には、PPC48の例外割込み調停論理48aは、構造的に規定された優先順位に従ってそれらの間で調停する。

【0048】レジスタR0-R63を含む上述した汎用レジスタはIFU26のレジスタファイル(OF)50に見出される。汎用レジスタ各々の広さないし大きさは64ビットである。OF50の制御はPPC48による。また、汎用レジスタR63は64ビット固定(「0」)である。

【0049】DEC44のモードBトランスレータ44aはモードB命令をモードA命令のシーケンスに変換することに責任がある。変換された命令は、次に、復号するためにDECのモードAデコーダ44bに伝達される。モードB変換に関して、DECはFE42の命令バッファ42aの底から16ビットを参照し、モードB命令をエミュレートするために1サイクル当たり1つのモードA命令を発出する。モードA命令は、FE42のマルチプレクサ43へ、次にモードAデコーダ44bまでルートバックされる。変換状態は、モードBエミュレートシーケンスの生成を制御するために、DEC44内に維持される。全てのエミュレート命令が生成されるとき、DEC44は、次のモードB命令にシフトすることをFE42に通知する。この場合のシフト位置は命令バッファ42aの最上ビットから16ビット又はバッファの下から16ビットである。

【0050】図4はFE42及びDEC44を更に詳細に示す。図4に示すように、命令バッファ(IB)42aはICCからフェッチされた命令を受け取る。命令は、IB42aから引っ張られ、動作のモード(即ち、モードB命令が使用中であって、モードA命令によってエミュレート中であるか、又は、モードA命令のみが使用中であるか)に応じて、DEC44のモードBトランスレータ44a及びモードAプリデコーダ44cに供給される。予め複号済みのモードA命令(モードAで作動

中であれば)、又は、モードBトランスレータからのモードA命令(モードBで作動中であれば)は、モードAデコーダ44bへ供給するためにマルチプレクサ43によって選定される。モードAプリデコーダは、32ビット命令信号に幾らかのプリデコード信号を加えた信号を生成する。モードBトランスレータは、モードA32ビット命令信号にデコード復号信号を加えた信号も生成し、この信号は、モードA命令が供給済みであれば、モードAプリデコーダによって生成されるプリデコード信号をエミュレートする。

【0051】FE42は、何の実行モードが存在するか、即ち、実行されているのはモードA命令であるか、又は、実行するためにモードA命令に変換されているのはモードB命令であるかを表示するために設定されるモードラッチ42bを含む。モードラッチ42bはマルチプレクサ43を制御する。理解されるように、本発明によれば、命令実行のモードは、分岐命令の目標アドレスの最下位ビット(LSB)によって決定される。モードA環境において作動中である時には、「0」にセットされた目標命令のアドレスのLSBを持ったモードAの無条件分岐命令(BLINK)を用いてモードBへの切り替えが実施される。モードBからモードAへの切り替えは、「1」にセットされたLSBを持つ目標アドレスを用いて、数個のモードB分岐命令によって開始される。

【0052】「delay slot present」(遅延スロット在り)ラッチ42cはFE42内に存在する。DSP42cは、変換され、エミュレートされ、分岐が実施可能になる以前に実行されるべき遅延スロット命令が変換中のモードB分岐命令に後続することを表示するように、DEC44のモードBトランスレータ44aからの信号によって設定される。遅延スロット命令が変換のためにモードBトランスレータ44aに送られるとき、DSP42eは、FE42によってリセットされる。

【0053】図4は、モードBトランスレータ44a、モードAデコーダ44b、及び、モードAプリデコーダ44cを含むDEC44を示す。モードB命令は、FE42から発出され、緩衝され、モードBトランスレータ44a、即ち、各モードB命令に対して1つ又は複数のモードA命令を生成する状態マシン実現回路に供給される。トランスレータ44aによって生成されたモードA命令はマルチプレクサ回路43を通過し、緩衝された後で、モードAデコーダ44bに供給される。次に、復号済み命令(即ち、オペランド、命令信号、等々)は実行ユニットに伝達される。

【0054】プロセッサエレメントの演算性能は分岐の効率に高度に依存する。従って、制御流れ機構は、低ペナルティ分岐をサポートするように設計されている。これは、CPUに当該分岐目標について通知する準備目標(PT)命令を、流れに対する制御、おそらく条件つき

で当該分岐目標に対する制御を起こさせる分岐命令から分離することによって、本発明により達成される。この技法は、ハードウェアが、多くのサイクル数だけ先立って、分岐目標について通知されることを可能にし、分岐が実施される場合には、ハードウェアが命令の現行シーケンスから目標シーケンスへの円滑な遷移を準備することを可能にする。また、この段階において、分岐は比較演算の包括的セットを符号化するのに十分な空間を持つので、この配置構成は、分岐命令における一層多くの融通性を可能にする。これらは、1つの単一命令内に比較と分岐両方のオペレーションを含むので、折量み比較分岐と呼ばれる。

【0055】モードB命令セットアーキテクチャに用いられるレジスタは一般に広さが32ビットであり、その個数(例えば16)はモードA命令セットアーキテクチャに使用されるレジスタの個数(64個、広さは各64ビット)よりも少ないものであることもあり得る。従って、モードB命令実行用汎用レジスタは、OF50のモードA汎用レジスタ16個の下から32ビットにマップされる。更に、上述したように、符号付きエクステンションが用いられる。即ち、オペランド又はモードB命令の他の表現式がOF50の汎用レジスタに書き込まれる時、上側ビット位置(32-63)にコピーされた最上位ビット(ビット位置31)を伴った下側ビット(ビット位置0-31)に書き込まれる。更に、モードB命令セットに用いられる状態レジスタの状態は、モードAアーキテクチャの特定レジスタビットにマップされる。

【0056】図5に示すマッピングの例は、早期開発されたモードB命令セットアーキテクチャの状態、及び、それがマップされた、モードAアーキテクチャ状態を示す。特定の命令セットに関して、本技術分野における当業者は、マッピングが利用可能な資源に依存することを認識するはずである。従って、ここに示す特定のマッピングは、当該マッピングに関連する命令セットアーキテクチャに依存する事例にすぎない。図5は、モードA状態(右端欄)へのモードB状態(左端欄)のマッピングを示す。例えば、モードBアーキテクチャのプログラムカウンタ状態は、モードAアーキテクチャのプログラムカウンタの低位ビット位置にマッピングされる。

【0057】レジスタマッピングに加えて、種々フラグの状態もマップされる。図5に示すように、1ビットフラグは、モードAアーキテクチャの一般レジスタの1つ又はもう一方の特定ビット位置にマップされる。従って、例えば、モードA T、S、M、Q状態/フラグは、汎用レジスタR25(ビット位置0)、及び、SR46b(フィールドS、M、Q)にそれぞれマップされる。

【0058】広さ32ビットのモードA命令は4バイト境界上に記憶され、モードB命令は4バイト又は2バイト境界のどちらかに記憶される。従って、少なくとも2

ビット (LSBおよびLSB+1) はアドレッシング用として未使用であり、オペレーションのモードを識別するために利用可能である。モードAとモードB命令実行の間のスイッチングは、分岐の目標アドレスの2つのLSBを検出する分岐命令を用いて達成される。モードA命令の実行に際して、モードAオペレーションからモードBオペレーションへスイッチすることは無条件分岐アドレス (BLINK) のみが可能である。従って、オペレーションのモードは、モードAおよびB命令セットアーキテクチャに用いられるジャンプ命令の目標アドレスのLSBを用いて変更可能である。このビット位置における「0」はモードB目標命令を表示し、「1」はモードA目標命令を表示する。LSBは、モード表示にのみ用いられ、実際の目標アドレスに影響しない。

【0059】初期のモードB命令セットアーキテクチャは、分岐 (分岐) オペレーションに関して生じるペナルティを軽減するために遅延スロット機構を利用した。遅延スロットは分岐命令に密着して後続する命令であり、当該分岐がプログラム流において遷移を引き起こす (又は、引き起こさない) ことができる以前に実行される。既に示したように、当該分岐に充分先立って目標アドレスレジスタに当該分岐の目標アドレスをロードするために、一層円滑な遷移をPT命令によって実施可能である。ただし、遅延スロット付きモードB分岐命令のエミュレーションは当該遅延スロットについて解明しなければならない。従って、遅延スロット付きモードB分岐命令に遭遇した時には、モードAコードシーケンスは当該分岐を実施するが、目標命令は、当該分岐命令 (即ち、遅延スロット命令) に後続するモードB命令がエミュレートされて完了するまで実行されない。

【0060】図6は本発明の一態様のモードA及びモードB環境両方におけるPT命令の使用を含む用法、モードAスレッド58から更にコンパクトなモードBスレッドへのスイッチング (ステップ64-84)、及び、モードAスレッドへの復帰を示す。

【0061】本発明の理解は、分岐命令のオペレーションについての記述から最もよく実現可能である。

【0062】モードAからモードAへの分岐: 図4および6を参照することとし、モードA命令ストリーム58 (図6) が実行中である間に、BLINK命令を用いて無条件分岐が別のモードA命令ストリームに対して (即ち、モード切り替えなしに) 実施されるものと仮定する。PT命令は、復号するためにIB42aからBLINK命令がプルされる幾らか以前に、8個の目標アドレスレジスタ46aの1つに、そこに分岐が実施されるべき目標アドレスをロードする (ステップ60)。その後で、BLINK命令は、IB42aの最上部に到達し、部分的復号のために、モードAプリデコード44cに送られ、次に、マルチプレクサ43を介してDEC44のモードAデコード44bまで送られる (ステップ6

2)。DEC44は、目標命令のアドレスを含む目標アドレスレジスタ46aの識別表示を有する復号済み情報をBR46に送る。

【0063】ステップ66において、BR46は、識別された目標アドレスレジスタ46aから目標アドレスを読み取り、それを、分岐コマンド信号と共にFE42に送る。続いて、BRは、分岐命令に後続する実行パイプライン内の全ての命令を無効化する。

【0064】一方、FE42は、ステップ68において、ICU27から目標命令をフェッチするために、BR46から受け取った目標アドレスを用いてフェッチリクエストをICC40へ発出する (図2)。ステップ70において、FE42は、目標アドレスのLSBをチェックする。LSBが「0」であれば、FEは、目標命令がモードB命令であることを知るはずである。ただし、この場合、目標命令はモードA命令であるので、LSBは「1」であり、モード変化は起こらない。ほぼ同時に、目標命令とそれに後続する命令の命令ストリームを受取る準備のために、IB42aの内容は無効化される。目標命令は、ICC40から受け取られると、IB42a内に置かれ、復号するために、そこからDEC44に送られ、ステップ72において、オペレーションは継続する。

【0065】モード切り替え: モードAからモードBへの分岐: ここで、モードA命令シーケンス内において、モードBシーケンスの更にコンパクトなコードへの切り替え (スイッチ) が行われるものと仮定する。この点で、目標アドレスのLSBの使用が役立つことになる。初めに、目標命令がモードB命令であることを表示するために「0」に設定されたLSBを有する目標アドレスを、PT命令が、目標アドレスレジスタ46aにロードすることをステップ60が確認することを除けば、ステップ60-68は、上述の場合と同じである。次に、モードA実行からモードB実行へのスイッチ用に用いられるBLINK分岐命令がDEC44に送られて、復号される (ステップ62)。BLINK命令を復号した後で、DEC44は、分岐用に使用するために、目標アドレスレジスタ46aの識別をBR46に送る。一方、BRは、識別済み目標アドレスレジスタ46aの内容を読み取り、分岐コマンド信号と共にそれをFE42に送り (ステップ66)、分岐命令に後続する実行パイプライン内のあらゆる命令を無効化する。FE42は、目標アドレスを用いてフェッチリクエストをICC40に送り、その代わりに、目標命令を受け取る (ステップ68)。更に、ステップ70において、FEは、目標アドレスの下位ビット (即ち、LSB) が「0」であることを検出し、モードBオペレーションを表示するようにモードラッチ42bを適宜設定することによって、その内部モード状態をモードAからモードBへ変える (ステップ76)。モードラッチ42bの出力は、モードBトラ

ンスレータ44aからモードAデコーダ44bへ命令を通信するようにマルチプレクサ43を制御する。

【0066】ここで、スイッチは完了する。次に、命令はモードBトランスレータへ送られ(ステップ78)、ここで、これらの命令はモードB命令をエミュレートするモードA命令に変換される。

【0067】モードBからモードBへの分岐:モードBにおいて作動中の期間における分岐は基本的に上述した場合と同じである。モードB分岐命令は、目標レジスタ46aに目標命令のアドレスをロードするPT命令を含むモードA命令のシーケンスに変換され、分岐を実行するために、モードA分岐命令(例えばBLINK分岐命令)がこれに続く。当該分岐が実施可能である以前に実行されなければならない分岐命令に後続する遅延スロット命令をモードB分岐命令が表示するならば例外である。遅延スロット命令がモードB分岐命令に後続しないならば、目標命令のアドレスを提供するためにPT命令によって先行された後、モードA分岐に関して既に概説したステップが実施される。

【0068】しかし、遅延スロット命令が存在するならば、モードBトランスレータ44aは、分岐命令を復号し、それが遅延スロット命令の存在を表示することを注記することによって、FEに対して遅延スロットが存在することを表示するFEにおけるラッチ42cをセットするためにDS.d信号をFE42に表明する。BR46が分岐目標アドレスをFE42に送る時には、FE42は、遅延スロット命令を除くIB42aの全ての内容を無効化する。FEは、目標命令をフェッチすることをICC40に要求し、遅延スロット命令が未だDEC44に転送されていないければ、それを受け取ると、それを遅延スロット命令の背後に置く。また、FE42は、分岐目標アドレスのLSBを調べる。LSBが「0」であるならば、モードビット42bは変更されない状態のままである。

【0069】遅延スロット命令は、モードBトランスレータに供給され、それをエミュレートするモードA命令を生成するために変換され、次に、FE42はDSP42cを「0」にリセットする。遅延スロット命令のエミュレーションが完了すると、分岐目標命令はモードBトランスレータへ供給される。

【0070】モード切り替え:モードBからモードAへの分岐:再び、モードB命令が実行中であっても、実施される初期ステップは基本的に上述したものと同じである。モードB分岐命令は、モードBトランスレータによって、目標アドレスレジスタにモードA目標命令の目標

アドレス(LSBは「1」に設定)をロードするためのPT命令を有するモードA命令シーケンスを生成するように変換される。また、モードB分岐命令がそれに後続する遅延スロット命令を持つならば、モードBトランスレータはDS.d信号をFEに発出し、遅延スロット命令が存在することを表示するためにFEのDSPラッチ42cを設定する。BRは、目標がモードA命令であることを表示するために、そのLSBが「1」に設定された目標アドレスの内容を読み取り、それをFE42に送る。次に、BR46は、パイプライン内に偶然存在するならば遅延スロット命令のエミュレーションを除き、分岐命令に後続するパイプライン内の全ての命令を無効化する。

【0071】目標アドレスを受け取ると、FE42は、目標アドレスを用いてICC40にフェッチリクエストを発出し、遅延スロット命令を除くIB42aの内容を無効化する。遅延スロットが変換された後で、モードAオペレーションを表示するようにモードラッチを設定することによりFE42はそのモード状態を変える。ここにおいて、目標命令を含むIB42aからの更なる全ての命令はマルチプレクサ43によってモードAアプリデコーダ44cに経路指定される。

【0072】

【発明の効果】本発明によれば、以下の優れた効果が得られる。

【0073】(1).16ビットコード及び32ビットコードの両方を実行する能力は、プロセッサが普通のタスクのためにコンパクトな16ビットコードを使用することを可能にする。

【0074】(2).両メモリスパースの節減および当該節減を伴った他の利点(例えば、更に小さいメモリ、節減された電力消費等)を可能にする。更に複雑なタスクが必要とされるときは、32ビットコードが使用できる。

【0075】(3).初期に設計された16ビット命令セットアーキテクチャを実行する能力は、当該初期設計においてなされた投資の保持を可能にする互換性を提供する。

【0076】(4).PT命令は、分岐に関する事前通知を提供することにより、分岐命令の遂行におけるより多くの融通性を可能にする。

【0077】以下に本発明の付録A、Bを示す。

【0078】付録A

モードBデータ転送命令エミュレーションシーケンス

【0079】

【表1】

モードB命令	モードA命令シーケンス	入	出
MOV #imm,Rn 1110 nnnn slll ddd	1 movl #imm,Rn		
MOV.W @(disp,PC),Rn 1001 nnnn dddd dddd	1 mova.w disp,R32 2 ld.w R32,#0,Rn		
MOV.L @(disp,PC),Rn 1101 nnnn dddd dddd	1 mova.l disp,R32 2 ld.l R32,#0,Rn		
MOV Rm,Rn 0110 nnnn mmmm 0011	1 addl Rm,#0,Rn		
MOV.B Rm,@Rn 0010 nnnn mmmm 0000	1 stb Rn,#0,Rm		
MOV.W Rm,@Rn 0010 nnnn mmmm 0001	1 stw Rn,#0,Rm		
MOV.L Rm,@Rn 0010 nnnn mmmm 0010	1 stl Rn,#0,Rm		
MOV.B @Rm,Rn 0110 nnnn mmmm 0000	1 ld.b Rm,#0,Rn		
MOV.W @Rm,Rn 0110 nnnn mmmm 0001	1 ld.w Rm,#0,Rn		
MOV.L @Rm,Rn 0110 nnnn mmmm 0010	1 ld.l Rm,#0,Rn		
MOV.B Rm,@-Rn 0010 nnnn mmmm 0100	1 stb Rn,#-1,Rm 2 addl.l Rn,#-1,Rn		
MOV.W Rm,@-Rn 0010 nnnn mmmm 0101	1 stw Rn,#-1,Rm 2 addl.l Rn,#-2,Rn		
MOV.L Rm,@-Rn 0010 nnnn mmmm 0110	1 stl.w Rn,#-1,Rm 2 addl.l Rn,#-4,Rn		
MOV.B @Rm+,Rn 0110 nnnn mmmm 0100	1 ld.b Rm,#0,Rn 2 if (m!=n) addl.l Rm,#1,Rm		
MOV.W @Rm+,Rn 0110 nnnn mmmm 0101	1 ld.w Rm,#0,Rn 2 if (m!=n) addl.l Rm,#2,Rm		
MOV.L @Rm+,Rn 0110 nnnn mmmm 0110	1 ld.l Rm,#0,Rn 2 if (m!=n) addl.l Rm,#4,Rm		
MOV.B R0,@(disp,Rm) 1000 0000 mmmm dddd	1 stb Rn,disp,R0		
MOV.W R0,@(disp,Rm) 1000 0001 mmmm dddd	1 stw Rn,disp,R0		
MOV.L Rm,@(disp,Rn) 0001 nnnn mmmm dddd	1 stl Rn,disp,Rm		
MOV.B @(disp,Rm),R0 1000 0100 mmmm dddd	1 ld.b Rm,disp,R0		
MOV.W @(disp,Rm),R0 1000 0101 mmmm dddd	1 ld.w Rm,disp,R0		

【0080】

* * 【表2】

MOV.L @(disp,Rn),Rn 0101 nnnn mmmmm dddd	1 ld.l Rn,disp,Rn		
MOV.B Rn,@(R0,Rn) 0000 nnnn mmmmm 0100	1 stx.b Rn,R0,Rn		
MOV.W Rn,@(R0,Rn) 0000 nnnn mmmmm 0101	1 stx.w Rn,R0,Rn		
MOV.L Rn,@(R0,Rn) 0000 nnnn mmmmm 0110	1 stx.l Rn,R0,Rn		
MOV.B @(R0,Rn),Rn 0000 nnnn mmmmm 1100	1 ldx.b Rn,R0,Rn		
MOV.W @(R0,Rn),Rn 0000 nnnn mmmmm 1101	1 ldx.w Rn,R0,Rn		
MOV.L @(R0,Rn),Rn 0000 nnnn mmmmm 1110	1 ldx.l Rn,R0,Rn		
MOV.B R0,@(disp,GBR) 1100 0000 dddd dddd	1 stx.b R27,disp,R0		
MOV.W R0,@(disp,GBR) 1100 0001 dddd dddd	1 stx.w R27,disp,R0		
MOV.L R0,@(disp,GBR) 1100 0010 dddd dddd	1 stx.l R27,disp,R0		
MOV.B @(disp,GBR),R0 1100 0100 dddd dddd	1 ld.b R27,disp,R0		
MOV.W @(disp,GBR),R0 1100 0101 dddd dddd	1 ld.w R27,disp,R0		
MOV.L @(disp,GBR),R0 1100 0110 dddd dddd	1 ld.l R27,disp,R0		
MOVA @(disp,PC),R0 1100 0111 dddd dddd	1 mova.l disp,R0		
MOVT Rn 0000 nnnn 0010 1001	1 andl R25,#1,Rn		
SWAP.B Rm,Rn 0110 nnnn mmmmm 1000	1 byte rev Rm,R32 2 shrl Rm,#16,Rn 3 mextr6 R32,Rn,Rn	Rm	
SWAP.W Rm,Rn 0110 nnnn mmmmm 1001	1 mperm.w Rm,#1,R32 2 addl.l R32,#0,Rn		
XTRACT Rm,Rn 0010 nnnn mmmmm 1101	1 shll.l Rm,#16,R32 2 shrl.l Rn,#16,Rn 3 or Rn,R32,Rn		

算術命令エミュレーションシーケンス

モード B 命令	モード A 命令シーケンス	入	出
ADD Rm,Rn 0011 nnnn mmmmm 1100	1 add.l Rm,Rn,Rn		
ADD #imm,Rn 0111 nnnn slll iiii	1 add.l Rn,#imm,Rn		
ADDC Rm,Rn 0011 nnnn mmmmm 1110	1 addz.l Rm,R63,R32 2 addz.l Rn,R63,Rn		

【0081】

* * 【表3】

25

26

	3 add Rn,R32,Rn 4 add Rn,R25,Rn 5 shlr Rn,#32,R25 6 addl.l Rn,#0,Rn		
ADDV Rn,Rn 0011 nnnn yccc 1111	1 add Rn,Rn,R32 2 add.l Rn,Rn,Rn 3 cmpne Rn,R32,R25	Rn Rn	
CMP/EQ #imm,R0 1000 1000 slll 00	1 movl #imm,R32 2 cmpeq R0,R32,R25	R0	
CMP/EQ Rn,Rn 0011 nnnn mmmn 0000	1 cmpeq Rn,Rn,R25	Rn Rn	
CMP/HS Rn,Rn 0011 nnnn mmmn 0010	1 cmpgeu Rn,Rn,R25	Rn Rn	
CMP/GE Rn,Rn 0011 nnnn mmmn 0011	1 cmpge Rn,Rn,R25	Rn Rn	
CMP/HI Rn,Rn 0011 nnnn mmmn 0110	1 cmpgtu Rn,Rn,R25	Rn Rn	
CMP/GT Rn,Rn 0011 nnnn mmmn 0111	1 cmpgt Rn,Rn,R25	Rn Rn	
CMP/PZ Rn 0100 nnnn 0001 0001	1 cmpge Rn,R63,R25	Rn	
CMP/PL Rn 0100 nnnn 0001 0101	1 cmpgt Rn,R63,R25	Rn	
CMP/STR Rn,Rn 0010 nnnn mmmn 1100	1 mcmpeq.b Rn,Rn,R32 2 addz.l R32,R63,R32 3 cmpgtu R32,R63,R25		
DIVS Rn,Rn 0010 nnnn mmmn 0111	1 xor Rn,Rn,R32 2 ori Rn,#0,R33 3 ori Rn,#0,R33 4 shlr.l R32,#31,R32 5 xori R32,#1,R25 Q.d = PPF_EX2[63] 6 nop M.d = PPF_EX2[63] #1.5 dec br sr update = 1		
DIVU 0000 0000 0001 1001	1 movl #0,R25 Q.d = 0 M.d = 0 #1.5 dec br sr update = 1		
DIV1 Rn,Rn 0011 nnnn mmmn 0100	1 addz.l Rn,R63,R32 2 ori Rn,#0,R33 oldQ.d = Q.q 3 shll.l Rn,#1,Rn 4 addz.l Rn,R25,Rn 5 if (oldQ.q == M.q) sub Rn,R32,Rn else add Rn,R32,Rn Q.d = PPF_EX2[63] 6 addl.l Rn,#0,Rn oldQ.d = Q.q ^ M.q		

【0082】

* * 【表4】

	7 nop		
	8 nop		
	$Q_d = oldQ_q \wedge PPF_EX2[32]$		
	9 if (Q.q == 1) movl #1, R25		
	else movl #0, R25		
	#1.5 dec br sr update = 1		
DT Rn	1 addl.l Rn, #-1, Rn		
0100 nnnn 0001 0000	2 cmpeq Rn, R63, R25		
DMULS.L Rm, Rn	1 muls.l Rm, Rn, R24		
0011 nnnn mmmmm 1101			
DMULU.L Rm, Rn	1 mulu.l Rm, Rn, R24		
0011 nnnn mmmmm 0101			
EXTS.B Rm, Rn	1 shlll Rm, #56, Rn		
0110 nnnn mmmmm 1110	2 sharl Rn, #56, Rn		
EXTS.W Rm, Rn	1 shlll Rm, #48, Rn		
0110 nnnn mmmmm 1111	2 sharl Rn, #48, Rn		
EXTU.B Rm, Rn	1 andl Rm, #255, Rn		
0110 nnnn mmmmm 1100			
EXTU.W Rm, Rn	1 shlll Rm, #48, Rn		
0110 nnnn mmmmm 1111	2 shrl Rn, #48, Rn		
MULL Rm, Rn	1 mulu.l Rm, Rn, R32		
0000 nnnn mmmmm 0111	2 shrl R24, #32, R24		
	3 mshfll.l R32, R24, R24		
MULS.W Rm, Rn	1 mmulo.wl Rm, Rn, R32		
0010 nnnn mmmmm 1111	2 shrl R24, #32, R24		
	3 mshfll.l R32, R24, R24		
MULU.W Rm, Rn	1 shlll Rm, #48, R32		
0010 nnnn mmmmm 1110	2 shlll Rn, #48, R32		
	3 shrl R32, #48, R32		
	4 shrl R33, #48, R32		
	5 mulu.l R32, R33, R32		
	6 shrl R24, #32, R24		
	7 mshfll.l R32, R24, R24		
NEG Rm, Rn	1 sub.l R63, Rm, Rn		
0110 nnnn mmmmm 1011			
NEGC Rm, Rn	1 addz.l Rm, R63, R32		Rm
0110 nnnn mmmmm 1010	2 sub R63, R25, Rn		
	3 sub Rn, R32, Rn		
	4 shrl Rn, #32, R25		
	5 addl.l Rn, #0, Rn		
SUB Rm, Rn	1 sub.l Rn, Rm, Rn		
0011 nnnn mmmmm 1000			
SUBC Rm, Rn	1 addz.l Rm, R63, R32		
0110 nnnn mmmmm 1010	2 addz.l Rn, R63, Rn		
	3 sub Rn, R25, Rn		
	4 sub Rn, R32, Rn		
	5 shrl Rn, #32, R25		
	6 addl.l Rn, #0, Rn		

【0083】付録B

ADD. L

説明: ADD. L命令は、R_aの下位32ビットをR_nの
下位32ビットに加算し、かつ符号拡張された32ビ
ットの結果をレジスタR_dに格納する。ビットR_a (32 *
40 【0084】
【表5】

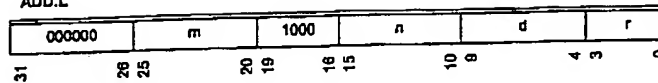
*FOR 32) およびR_n(32 FOR 32)は無視される。

操作:

29

30

ADD.L

ADD.L R_m, R_n, R_d

```

source1 ← SignExtend32( $R_m$ );
source2 ← SignExtend32( $R_n$ );
result ← SignExtend32(source1 + source2);
 $R_d$  ← Register(result);

```

【0085】ADDZ.L

*FOR 32) および R_n (32 FOR 32) は無視される。

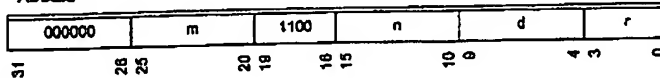
説明：ADDZ.L命令は、 R_m の下位32ビットを R_n の下位32ビットに加算し、かつ零拡張された32ビットの結果をレジスタ R_d に格納する。ビット R_m (32

操作：

【0086】

【表6】

ADDZ.L

ADDZ.L R_m, R_n, R_d

```

source1 ← ZeroExtend32( $R_m$ );
source2 ← ZeroExtend32( $R_n$ );
result ← ZeroExtend32(source1 + source2);
 $R_d$  ← Register(result);

```

【0087】ADDI.L

※視される。

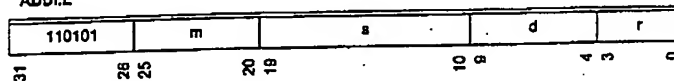
説明：ADDI.L命令は、 R_m の下位32ビットをインターネットミーディエイト（即値） s の符号拡張された値に加算し、かつ符号拡張された32ビットの結果をレジスタ R_d に格納する。ビット R_m (32 FOR 32) は無※

操作：

【0088】

【表7】

ADDI.L

ADDI.L R_m, s, R_d

```

source1 ← SignExtend32( $R_m$ );
source2 ← SignExtend10( $s$ );
result ← SignExtend32(source1 + source2);
 $R_d$  ← Register(result);

```

【0089】SHLLI.L

★操作：

説明：SHLLI.L命令は、 R_m の下位32ビットを S (FOR 5) だけ論理的に左シフトさせ、かつ符号拡張された32ビットの結果をレジスタ R_d に格納する。★

【0090】

【表8】

31

32

SHLL.L

110001	m	0000	s	d	r
31	28 25	20 19 16 15	10 9	4 3	0

SHLL.L R_m, s, R_d

```

source1 ← ZeroExtend32( $R_m$ );
source2 ← ZeroExtend5(SignExtend6(s));
result ← SignExtend32(source1 << source2);
 $R_d$  ← Register(result);

```

【0091】SHLRI.L

* 操作:

説明: SHLRI.L 命令は、 R_a の下位32ビットを
 $S(0 \text{ FOR } 5)$ だけ論理的に右シフトさせ、かつ符号拡張
 された32ビットの結果をレジスタ R_d に格納する。 *

【0092】

【表9】

SHLRI.L

110001	m	0010	s	d	r
31	28 25	20 19 16 15	10 9	4 3	0

SHLRI.L R_m, s, R_d

```

source1 ← ZeroExtend32( $R_m$ );
source2 ← ZeroExtend5(SignExtend6(s));
result ← SignExtend32(source1 >> source2);
 $R_d$  ← Register(result);

```

【0093】SHLLD.L

※ 操作:

説明: SHLLD.L 命令は、 R_a の下位32ビットを
 $R_n(0 \text{ FOR } 5)$ だけ論理的に左シフトさせ、かつ符号拡張
 された32ビットの結果をレジスタ R_d に格納する。 ※

【0094】

【表10】

SHLLD.L

000001	m	0000	n	d	r
31	28 25	20 19 16 15	10 9	4 3	0

SHLLD.L R_m, R_n, R_d

```

source1 ← ZeroExtend32( $R_m$ );
source2 ← ZeroExtend5( $R_n$ );
result ← SignExtend32(source1 << source2);
 $R_d$  ← Register(result);

```

【0095】SHLRD.L

★ 操作:

説明: SHLRD.L 命令は、 R_a の下位32ビットを
 $R_n(0 \text{ FOR } 5)$ だけ論理的に右シフトさせ、かつ符号拡張
 された32ビットの結果をレジスタ R_d に格納する。 ★

【0096】

40 【表11】

SHLRD.L

000001	m	0010	n	d	r
31	28 25	20 19 16 15	10 9	4 3	0

SHLRD.L R_m, R_n, R_d

```

source1 ← ZeroExtend32( $R_m$ );
source2 ← ZeroExtend5( $R_n$ );
result ← SignExtend32(source1 >> source2);
 $R_d$  ← Register(result);

```

33

34

【0097】SHARD.L

* 操作:

説明: SHARD.L 命令は、 R_n の下位32ビットを
 $R_n(0 \text{ FOR } 5)$ だけ算術的に右シフトさせ、かつ符号拡
 張された32ビットの結果をレジスタ R_d に格納する。 *

【0098】

【表12】

SHARD.L

000001	m	0110	n	d	r
31	25 23	20 19	18 15	12 8	4 3 0

SHARD.L R_m, R_n, R_d

```

source1 ← SignExtend32( $R_m$ );
source2 ← ZeroExtend5( $R_n$ );
result ← SignExtend32(source1 >> source2);
 $R_d$  ← Register(result);

```

【0099】LDHI.L

※ 操作:

説明: 不整合の、符号付けされたロング (長) ワードの
 上位部分をメモリから汎用レジスタにロードする。 ※

【0100】

【表13】

LDHI.L

110000	m	0110	s	d	r
31	25 23	20 19	18 15	12 8	4 3 0

LDHI.L R_m, s, R_d

```

base ← ZeroExtend32( $R_m$ );
offset ← SignExtend5(s);
address ← base + offset;
count ← (address & 0x3) + 1;
address ← ZeroExtend32(address & (~0x3));
count8 ← count * 8;
shift ← ZeroExtend5((~((base + offset) & 0x3)) * 8);
mem ← ZeroExtendcount8(MisalignedReadMemorycount8(address));
IF (isLittleEndian())
  result ← SignExtend32(mem << shift);
ELSE
  result ← ZeroExtend32(mem);
 $R_d$  ← Register(result);

```

【0101】LDLO.L

★ 操作:

説明: 不整合の、符号付けされたロング (長) ワードの
 下位部分をメモリから汎用レジスタにロードする。 ★

【0102】

【表14】

LDLO.L

110000	m	0010	s	d	r
31	25 23	20 19	18 15	12 8	4 3 0

LDLO.L R_m, s, R_d

```

base ← ZeroExtend32( $R_m$ );
offset ← SignExtend5(s);
address ← ZeroExtend32(base + offset);
count ← 4 - (address & 0x3);
count8 ← count * 8;
shift ← (address & 0x3) * 8;
mem ← ZeroExtendcount8(MisalignedReadMemorycount8(address));
IF (isLittleEndian())
  result ← ZeroExtend32(mem);
ELSE
  result ← SignExtend32(mem << shift);
 $R_d$  ← Register(result);

```

【0103】STHI.L

☆50☆説明: ロング (長) ワードの上位部分を汎用レジスタか

らメモリに不整合状態でストアする。

操作:

*【0104】

*【表15】

STHLL

111000	m	0110	s	y	r
31	28 27	26 25	24 23	22 21	20

STHLL R_m, s, R_y
<pre> base ← ZeroExtend₆₄(R_m); offset ← SignExtend₆(s); value ← ZeroExtend₃₂(R_y); address ← base + offset; count ← (address & 0x3) + 1; address ← ZeroExtend₆₄(address & (~0x3)); IF (IsLittleEndian()) start ← (4 - count) * 8; ELSE start ← 0; count8 ← count * 8; MisalignedWriteMemory_{count8}(address, value, start FOR count8 >); </pre>

【0105】STLO.L

※操作:

説明: ロング (長) ワードの下位部分を汎用レジスタか

【0106】

らメモリに不整合状態でストアする。

※【表16】

STLO.L

111000	m	0010	s	y	r
31	28 27	26 25	24 23	22 21	20

STLO.L R_m, s, R_y
<pre> base ← ZeroExtend₆₄(R_m); offset ← SignExtend₆(s); value ← ZeroExtend₃₂(R_y); address ← ZeroExtend₆₄(base + offset); count ← 4 - (address & 0x3); IF (IsLittleEndian()) start ← 0; ELSE start ← (4 - count) * 8; count8 ← count * 8; MisalignedWriteMemory_{count8}(address, value, start FOR count8 >); </pre>

【図面の簡単な説明】

★すフローチャートである。

【図1】本発明を実現するように構成されたプロセッサエレメントを用いた処理システムを概略的に示すブロック図である。

【図2】図1に示すプロセッサエレメントの命令フェッチユニット (IFU) を示すブロック図である。

【図3】図2に示す分岐ユニットに含まれる状態レジスタのレイアウトを示す図である。

【図4】図2に示すデコーダ (DEC) を示すブロック図である。

【図5】1つの命令セットアーキテクチャから別の命令セットアーキテクチャへの状態マッピングを示す図である。

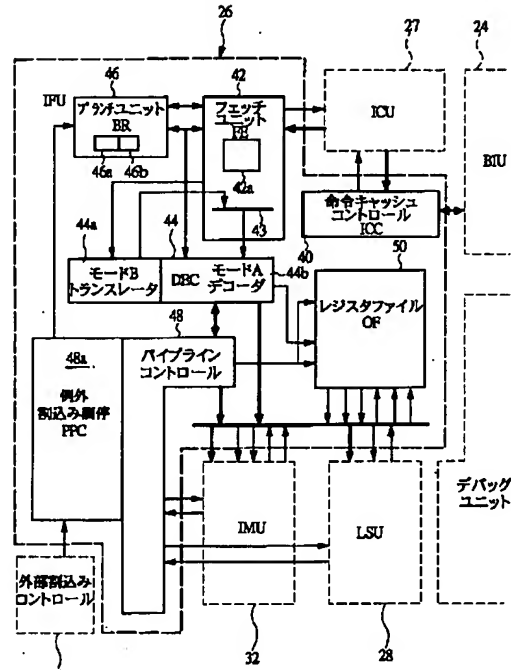
【図6】命令の流れを制御する本発明のアスペクトを示★

【符号の説明】

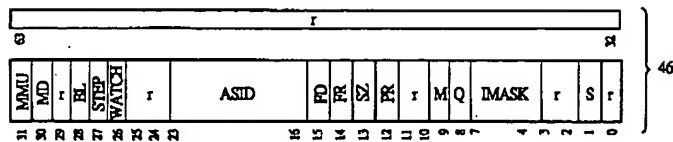
- 12 プロセッサエレメント
- 14 直接メモリアクセスユニット
- 20 システムバス
- 24 BIU: バスインタフェースユニット
- 26 IFU: 命令流ユニット
- 27 ICU: 命令キャッシュユニット
- 30 DCU: データキャッシュユニット
- 40 ICC: 命令キャッシュコントロール
- 42 FE: フェッチユニット
- 44 DEC: 復号ユニット
- 48 PPC: バイプラインコントロール
- 50 レジスタファイル

【図2】

2

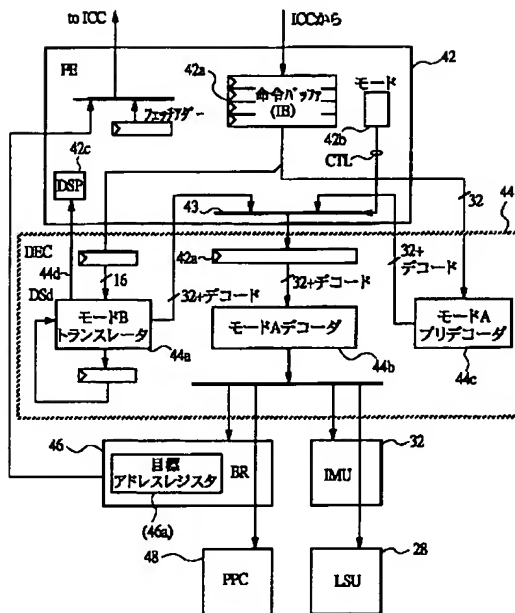


3



【図4】

図 4



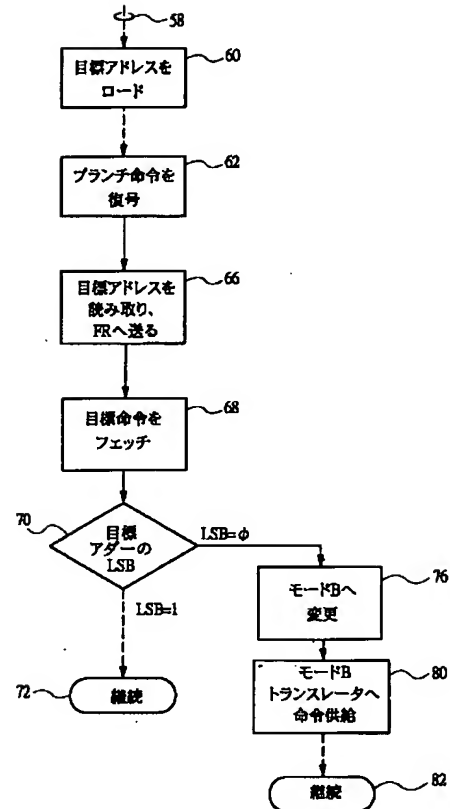
【図5】

図 5

モードB状態	記 述	モードA状態
PC	プログラムカウンタ	PCの下側32ビット
Ri, i = [0, 15]	モードB汎用レジスタ	Riの下側32ビット, i = [0, 15]
FR	手順リンクレジスタ	R18の下側32ビット
GBR	大域ベースレジスタ	R27の下側32ビット
MACL	乗算累積ロー	R24の下側32ビット
MACH	乗算累積ハイ	R24の上側32ビット
T	条件コードフラグ	R25のビット0
S	乗算累積飽和フラグ	SR.S
M	ディバイドステップMフラグ	SR.M
Q	ディバイドステップQフラグ	SR.Q

【図6】

図 6



フロントページの続き

(72)発明者 マーク・デバージ
 アメリカ合衆国、カリフォルニア州
 95014、クペルティノ、スイート 371、ス
 ティーブンス・クリーク・ブルバード
 5636

(72)発明者 セバスチャン・ハブルジ・ジースラー
 アメリカ合衆国、カリフォルニア州
 95118、サンノゼ、ジョセフ・レーン
 5315

(72)発明者 カナド・ロイ
アメリカ合衆国、カリフォルニア州
95054、サンタクララ、ナンバー111、パー
ク・ビュー・ドライブ 610

(72)発明者 アンドリュー・クレイグ・スタルジス
イギリス国、バス BA1 7RT、バス
フォード、パンプ・レーン 3
(72)発明者 ブラセンジット・ビスワス
アメリカ合衆国、カリフォルニア州
95070、サラトガ、パンパス・コート
20167